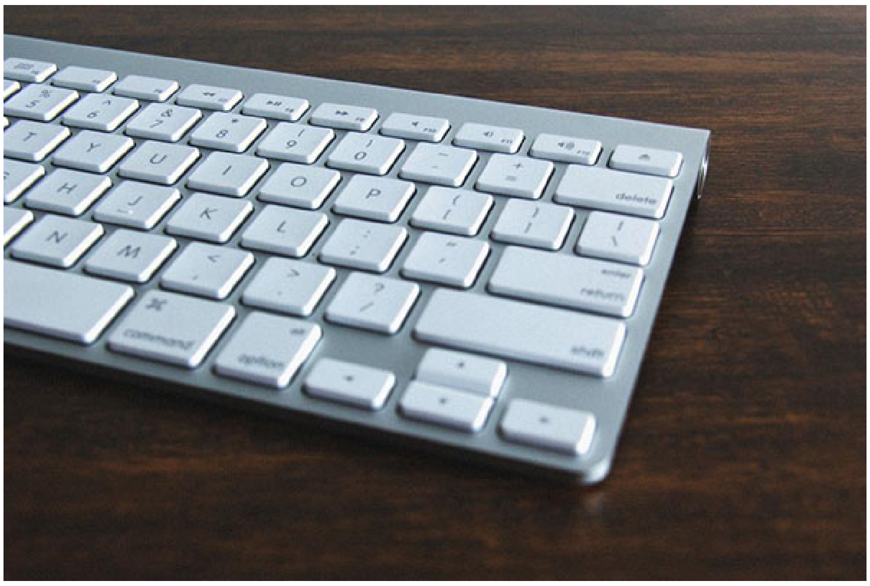
在 Swift 中实现 NS_OPTIONS

2015-08-17 14:20 编辑: lansekuangtu 分类: Swift 来源: 南峰子的技术博客

Swift NS_OPTIONS



从Xcode 4.5以后，我们在Objective-C中使用NS_ENUM和NS_OPTIONS来定义一个枚举，以替代C语言枚举的定义方式。其中NS_ENUM用于定义普通的枚举，NS_OPTIONS用于定义选项类型的枚举。

而到了Swift中，枚举增加了更多的特性。它可以包含原始类型(不再局限于整型)以及相关值。正是由于这些原因，枚举在Swift中得到了更广泛的应用。在Foundation中，Objective-C中的NS_ENUM类型的枚举，都会自动转换成Swift中enum，并且更加精炼。以Collection View的滚动方向为例，在Objective-C中，其定义如下：

```
typedef NS_ENUM(NSInteger, UICollectionViewScrollDirection) {
  UICollectionViewScrollDirectionVertical,
  UICollectionViewScrollDirectionHorizontal
};
```

而在Swift中，其定义如下：

```
enum UICollectionViewScrollDirection : Int {
  case Vertical
  case Horizontal
}
```

精练多了吧，看着舒服多了，还能少码两个字。我们自己定义枚举时，也应该采用这种方式。

不过对于Objective-C中NS_OPTIONS类型的枚举，Swift中的实现似乎就没有那么美好了。

我们再来对比一下UICollectionViewScrollPosition的定义吧，在Objective-C中，其定义如下：

```
typedef NS_OPTIONS(NSUInteger, UICollectionViewScrollPosition) {
    UICollectionViewScrollPositionNone                 = 0,
    // The vertical positions are mutually exclusive to each other, but are bitwise or-able with th
    // Combining positions from the same grouping (horizontal or vertical) will result in an NSInva
    UICollectionViewScrollPositionTop                  = 1 << 0,
    UICollectionViewScrollPositionCenteredVertically   = 1 << 1,
    UICollectionViewScrollPositionBottom               = 1 << 2,
    // Likewise, the horizontal positions are mutually exclusive to each other.
    UICollectionViewScrollPositionLeft                 = 1 << 3,
    UICollectionViewScrollPositionCenteredHorizontally = 1 << 4,
    UICollectionViewScrollPositionRight                = 1 << 5
};
```

而在Swift 2.0中，其定义如下：

```
struct UICollectionViewScrollPosition : OptionSetType {
    init(rawValue: UInt)
    static var None: UICollectionViewScrollPosition { get }
    // The vertical positions are mutually exclusive to each other, but are bitwise or-able with th
    // Combining positions from the same grouping (horizontal or vertical) will result in an NSInva
    static var Top: UICollectionViewScrollPosition { get }
    static var CenteredVertically: UICollectionViewScrollPosition { get }
    static var Bottom: UICollectionViewScrollPosition { get }
    // Likewise, the horizontal positions are mutually exclusive to each other.
    static var Left: UICollectionViewScrollPosition { get }
    static var CenteredHorizontally: UICollectionViewScrollPosition { get }
    static var Right: UICollectionViewScrollPosition { get }
}
```

额，光看代码，不看实现，这也是化简为繁的节奏啊。

为什么要这样做呢？Mattt给了我们如下解释：

Well, the same integer bitmasking tricks in C don’t work for enumerated types in Swift. An enum represents a type with a closed set of valid options, without a built-in mechanism for representing a conjunction of options for that type. An enum could, ostensibly, define a case for all possible combinations of values, but for n > 3, the combinatorics make this approach untenable.

意思是Swift不支持C语言中枚举值的整型掩码操作的技巧。在Swift中，一个枚举可以表示一组有效选项的集合，但却没有办法支持这些选项的组合操作(“&”、”|”等)。理论上，一个枚举可以定义选项值的任意组合值，但对于n > 3这种操作，却无法有效的支持。

为了支持类NS_OPTIONS的枚举，Swift 2.0中定义了OptionSetType协议【在Swift 1.2中是使用RawOptionSetType，相比较而言已经改进了不少】，它的声明如下：

```
/// Supplies convenient conformance to `SetAlgebraType` for any type
/// whose `RawValue` is a `BitwiseOperationsType`.  For example:
///
///     struct PackagingOptions : OptionSetType {
///       let rawValue: Int
///       init(rawValue: Int) { self.rawValue = rawValue }
///
///       static let Box = PackagingOptions(rawValue: 1)
///       static let Carton = PackagingOptions(rawValue: 2)
///       static let Bag = PackagingOptions(rawValue: 4)
///       static let Satchel = PackagingOptions(rawValue: 8)
///       static let BoxOrBag: PackagingOptions = [Box, Bag]
///       static let BoxOrCartonOrBag: PackagingOptions = [Box, Carton, Bag]
///     }
///
/// In the example above, `PackagingOptions.Element` is the same type
/// as `PackagingOptions`, and instance `a` subsumes instance `b` if
/// and only if `a.rawValue & b.rawValue == b.rawValue`.
protocol OptionSetType : SetAlgebraType, RawRepresentable {
    /// An `OptionSet`'s `Element` type is normally `Self`.
    typealias Element = Self
    /// Convert from a value of `RawValue`, succeeding unconditionally.
    init(rawValue: Self.RawValue)
}
```

从字面上来理解，OptionSetType是选项集合类型，它定义了一些基本操作，包括集合操作(union, intersect, exclusiveOr)、成员管理(contains, insert, remove)、位操作(unionInPlace, intersectInPlace, exclusiveOrInPlace)以及其它的一些基本操作。

作为示例，我们来定义一个表示方向的选项集合，通常我们是定义一个实现OptionSetType协议的结构体，如下所示：

```
struct Directions: OptionSetType {
    var rawValue:Int
    init(rawValue: Int) {
        self.rawValue = rawValue
    }
    static let Up: Directions = Directions(rawValue: 1 << 0)
    static let Down: Directions = Directions(rawValue: 1 << 1)
    static let Left: Directions = Directions(rawValue: 1 << 2)
    static let Right: Directions = Directions(rawValue: 1 << 3)
}
```

所需要做的基本上就是这些。然后我们就可以创建Directions的实例了，如下所示：

```
let direction: Directions = Directions.Left
if direction == Directions.Left {
    // ...
}
```

如果想同时支持两个方向，则可以如上处理：

```
let leftUp: Directions = [Directions.Left, Directions.Up]
if leftUp.contains(Directions.Left) && leftUp.contains(Directions.Up) {
    // ...
}
```

如果leftUp同时包含Directions.Left和Directions.Up，则返回true。

这里还有另外一种方法来达到这个目的，就是我们在Directions结构体中直接声明声明Left和Up的静态常量，如下所示：

```
struct Directions: OptionSetType {
    // ...
    static let LeftUp: Directions = [Directions.Left, Directions.Up]
    static let RightUp: Directions = [Directions.Right, Directions.Up]
    // ...
}
```

这样，我们就可以以如下方式来执行上面的操作：

```
if leftUp == Directions.LeftUp {
    // ...
}
```

当然，如果单一选项较多，而要去组合所有的情况，这种方法就显示笨拙了，这种情况下还是推荐使用contains方法。

总体来说，Swift中的对选项的支持没有Objective-C中的NS_OPTIONS来得简洁方便。而且在Swift 1.2的时候，我们还是可以使用”&“和”|”操作符的。下面这段代码在Swift 1.2上是OK的：

```
UIView.animateWithDuration(0.3, delay: 1.0, options: UIViewAnimationOptions.CurveEaseIn | UIViewAni
    // ...
}, completion: nil)
```

但到了Swift 2.0时，OptionSetType已经不再支持”&“和”|”操作了，因此，上面这段代码需要修改成：

```
UIView.animateWithDuration(0.3, delay: 1.0, options: [UIViewAnimationOptions.CurveEaseIn, UIViewAni
        // ...
}, completion: nil)
```

不过，慢慢习惯就好。

参考

- RawOptionSetType
- Exploring Swift 2.0 OptionSetTypes
- Notes from WWDC 2015: The Enumerated Delights of Swift 2.0 Option Sets
- 《100个Swift开发必备Tip》 — Tip 66. Options